



Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

CRT-2005-37

**ITERATED TABU SEARCH FOR THE
CAR SEQUENCING PROBLEM**

by

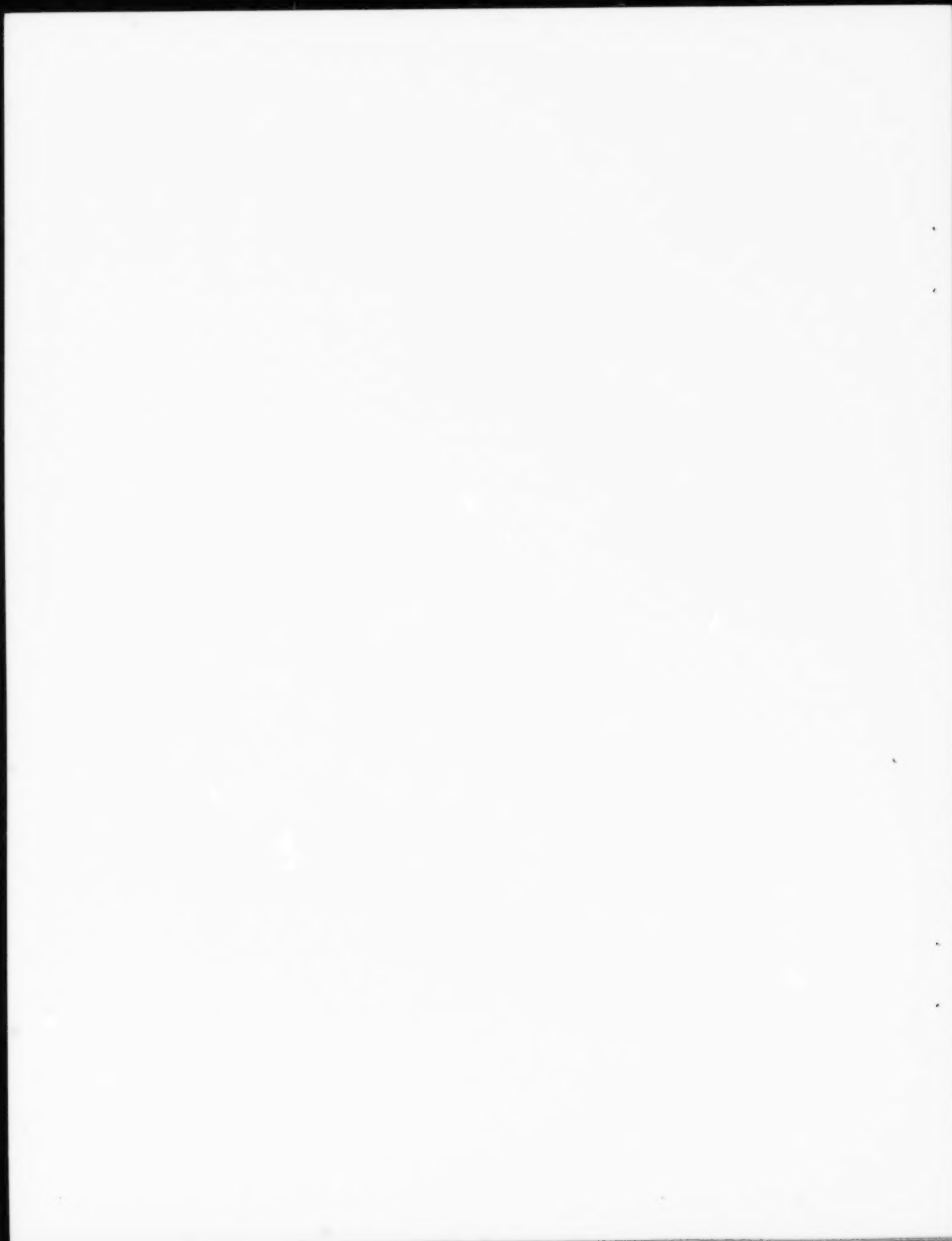
**Jean-François Cordeau
Gilbert Laporte
Federico Pasin**

December 2005

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone: (514) 343-7575 / Télécopieur: (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca





**PUBLICATION
CRT-2005-37**

**ITERATED TABU SEARCH FOR THE
CAR SEQUENCING PROBLEM**

by

**Jean-François Cordeau¹
Gilbert Laporte¹
Federico Pasin²**

This work was supported by a Strategic research grant provided by HEC Montréal, and by the Natural Sciences and Engineering Research Council of Canada (NSERC).

¹ Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7, and Canada Research Chair in Distribution Management, HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7 (cordeau, gilbert@crt.umontreal.ca)

² Department of Production and Operations Management, HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7 (Federico.Pasin@hec.ca)

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2005

Abstract

This paper introduces an iterated tabu search heuristic for the daily car sequencing problem in which a set of cars must be sequenced so as to satisfy requirements from the paint shop and the assembly line. The iterated tabu search heuristic combines a classical tabu search with perturbation operators that help escape from local optima. The resulting heuristic is flexible, easy to implement, and fast. It has produced very good results on a set of test instances provided by the French car manufacturer Renault.

Keywords: car sequencing problem, iterated local search, tabu search.

Résumé

Cet article présente une heuristique itérée de recherche avec tabous pour le problème quotidien d'ordonnancement de la production dans une usine d'assemblage de voitures. Dans ce problème, un ensemble de voitures doit être ordonné afin de répondre aux besoins des ateliers de peinture et d'assemblage. L'heuristique itérée combine la recherche avec tabous classique à des opérateurs de perturbation qui permettent d'échapper aux optima locaux. Cette heuristique est flexible, facile à implanter et rapide. Elle a produit de très bons résultats sur des instances fournies par le constructeur automobile français Renault.

Mots-clés: ordonnancement de voitures, recherche locale itérée, recherche avec tabous.

1 Introduction

In the *Car Sequencing Problem* (CSP), a set of cars must be sequenced to satisfy requirements associated with the various production stages of a car assembly process. We focus on the daily problem in which two particular stages must be considered: the paint shop and the assembly line. Because colour changes are costly and time consuming, cars of the same colour should appear consecutively in the sequence. Nevertheless, runs of cars of the same colour should not be too long since the painting equipment must be cleaned regularly and visual quality control becomes difficult when many cars of the same colour are produced in sequence. To ensure a smooth operation of the assembly line, cars must also be ordered so as to avoid the consecutive assembly of several cars possessing a given special feature, such as an opening roof or air conditioning.

Our aim is to present an iterated tabu search heuristic for the CSP. This work was initiated within the 2005 ROADEF Challenge which addressed the CSP encountered by the French car manufacturer Renault. After the competition, in which our team ranked seventh out of 55 registered teams, we further improved our approach which now yields very high quality solutions when compared to those of the winning team.

The terms of reference of the problem under consideration are described in the document provided to the challengers (Nguyen, 2005). Here, we briefly recall the problem and we introduce the notation used to describe our heuristic.

Let V be the set of all cars to be sequenced and $n = |V|$, the number of cars in this set. Let also R (with $m = |R|$) be the set of car features requiring special attention. With each such feature $r \in R$ is associated a ratio n_r/p_r indicating that in any subsequence of p_r cars, there should be at most n_r cars possessing the given feature. Each car $v \in V$ is defined by a colour c_v and a binary attribute vector $a_v = (a_{v_1}, \dots, a_{v_m})$, where a_{v_r} is 1 if and only if car v possesses feature r . Violations of the ratio constraint for feature r must be computed for every car subsequence of length p_r . Hence, to compute the number of violations in day j , one must take into account the last $p_r - 1$ cars from the solution of day $j - 1$, which is considered as an input to the problem. The solution of day $j + 1$ is, however, not taken into consideration. As a result, subsequences starting with cars in positions $n - n_r + 1, \dots, n$ need not be considered.

The CSP consists of finding a minimum cost ordering of the cars such that the number of consecutive cars of the same colour does not exceed a preset value ℓ . This *paint constraint* is hard and cannot be violated. However, ratio constraints are soft constraints whose violations should be minimized. Ratio constraints fall into two categories: *priority ratio constraints* (PRCs) and *nonpriority ratio constraints* (NPRCs). The objective function considered in this work is hierarchical and contains three terms: the number of colour changes, the number of PRC violations, and the number of NPRC violations. One must always avoid satisfying an NPRC at the expense of a PRC. Hence, three possible hierarchies of the terms are relevant:

1. colour changes, PRCs, NPRCs;
2. PRCs, colour changes, NPRCs;
3. PRCs, NPRCs, colour changes.

To transform the hierarchical objective into a unique one, the three terms are assigned weights of different magnitudes. Let α , β and γ be the weights associated with the number of colour changes, the number of PRC violations and the number of NPRC violations, respectively. Let also $P \subseteq R$ and $N \subseteq R$ represent the subsets of features corresponding to

PRCs and NPRCs. The cost of a solution s is then equal to

$$f(s) = \alpha c(s) + \beta \sum_{r \in P} d_r(s) + \gamma \sum_{r \in N} d_r(s),$$

where $c(s)$ and $d_r(s)$ denote the number of colour changes and the number of violations of ratio constraint $r \in R$, respectively.

The problem consists of determining a permutation of the cars in the set V minimizing $f(s)$ and containing no subsequence of cars of the same colour exceeding the maximum length ℓ . In the problem considered for the ROADEF Challenge, the values of α , β and γ are either 10^6 , 10^3 or 1, depending on the hierarchy considered for the three objectives.

The CSP is a difficult combinatorial optimization problem which has recently received increased attention (Kis, 2004). With the exception of Hindi and Ploszajski (1994) who proposed a construction heuristic, most work on the CSP has focused on metaheuristics (see, e.g., Delaval et al., 1995; Gagné et al., 2005; Gottlieb et al., 2003; Gravel et al., 2005; Muhl et al., 2003; Perron and Shaw, 2004; Ponnambalam et al., 2003; Smith et al., 1996; Solnon, 2000). In particular, simulated annealing, genetic search, large neighbourhood search, and ant colony optimization approaches have been proposed.

In this paper, we introduce an iterated tabu search heuristic which has yielded good results on the test instances proposed by Renault within the 2005 ROADEF Challenge. This heuristic possesses several practical strengths. First, it is easy to implement as it relies on simple operators. Second, it is flexible in the sense that it can easily be adapted to handle different variants of the CSP. It can also start from infeasible solutions. Third, it is fast: on the test instances provided by Renault, it usually produces within 30 seconds solutions that are close to the best known solutions.

The remainder of the paper is organized as follows. The next section describes the iterated tabu search heuristic. Section 3 presents computational experiments on data provided by Renault. Conclusions follow in the last section.

2 Iterated Tabu Search Heuristic

We have developed an *Iterated Tabu Search* (ITS) heuristic for the CSP. In ITS (see, e.g., Misevicius, 2004), the search process alternates between diversification and intensification steps. Let $\hat{s}$ represent the starting solution for the ITS process. In each cycle, a diversification step is first performed by perturbing $\hat{s}$ to obtain a new solution s' . Intensification is then performed around s' by applying a classical tabu search heuristic to produce a new solution $\tilde{s}$. If $\tilde{s}$ satisfies an acceptance criterion, it replaces $\hat{s}$ and the next perturbation is performed from

that solution. Otherwise, the search returns to the previous solution $\hat{s}$. The perturbations are aimed at escaping from local optima and exploring other parts of the search space. Iterated tabu search can be seen as a special case of the general iterated local search paradigm described by Lourenço et al. (2002). The complete process can be summarized by the following pseudo-code:

1. $s_0 \leftarrow$ initial solution;
2. apply tabu search on s_0 to obtain an improved solution $\hat{s}$;
3. while the termination criterion is not satisfied
 - (a) apply a perturbation on $\hat{s}$ to obtain s' ,
 - (b) apply tabu search on s' to obtain a local optimum $\tilde{s}$,
 - (c) if $\tilde{s}$ satisfies the acceptance criterion, $\hat{s} \leftarrow \tilde{s}$;
4. return the best solution found s^* .

It is worth pointing out that the solution $\hat{s}$ perturbed in Step 3a of the algorithm is not necessarily the best solution found since the beginning of the search. Indeed, the acceptance criterion specified in Step 3c may allow solutions that do not improve upon the best known solution s^* but are sufficiently close in terms of cost.

In the next sections, we describe the main steps of the algorithm: initialization, tabu search, perturbation and acceptance criterion. Computational considerations for efficient neighbourhood evaluation are then discussed in the last section.

2.1 Initialization

Since the algorithm allows infeasible intermediate solutions (i.e., solutions violating the paint constraint), it can be initialized with any permutation of the elements of V . In practice, we obtain an initial solution s_0 by considering the cars in the order in which they are defined in the input file. The next section describes how infeasible solutions are handled in the course of the search process.

2.2 Tabu search

Tabu search explores the solution space by moving at each iteration from the current solution s to the best solution in its neighbourhood $N(s)$. Since the solution may deteriorate between two successive iterations, an anti-cycling mechanism is put in place by declaring tabu some

attributes of recently visited solutions. We now explain how tabu search is adapted to our problem.

2.2.1 Relaxation mechanism

Infeasible solutions are allowed throughout the search by relaxing the paint constraint and by penalizing its violations in the objective function. The objective function $f(s)$ is thus replaced with a function $g(s) = f(s) + \zeta t(s)$ where $t(s)$ is the number of violations of the paint constraint in s and ζ is a positive parameter. The parameter ζ is initially set equal to 1 and self-adjusts in the course of search to allow a mix of feasible and infeasible solutions: at each iteration, the value of ζ is multiplied by 2 if the current solution is infeasible, and divided by 2 otherwise.

2.2.2 Solution neighbourhood

A solution is represented by an n -dimensional vector giving the position of each car in the sequence. For each position i with $1 \leq i \leq n$, let $v_i \in V$ denote the car assigned to position i in the current solution s . Two types of exchanges are used within tabu search to define the neighbourhood $N(s)$:

1. insert in position j the car v_i currently assigned to position $i \neq j$;
2. swap the cars v_i and v_j currently assigned to positions i and $j \neq i$.

In exchanges of type 1 (see Figure 1), v_{i-1} and v_{i+1} become consecutive after the exchange (unless $i = n$ in which case v_{i-1} becomes the last car in the sequence), and v_i is inserted between v_j and v_{j+1} . In exchanges of type 2 (see Figure 2), v_i and v_j are swapped and all other cars remain in the same position after the exchange.

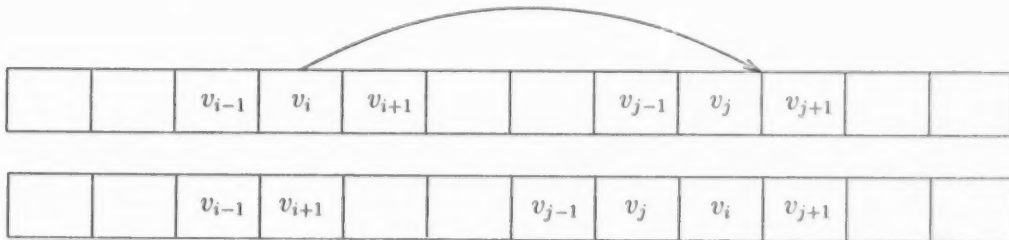


Figure 1: Exchange of type 1

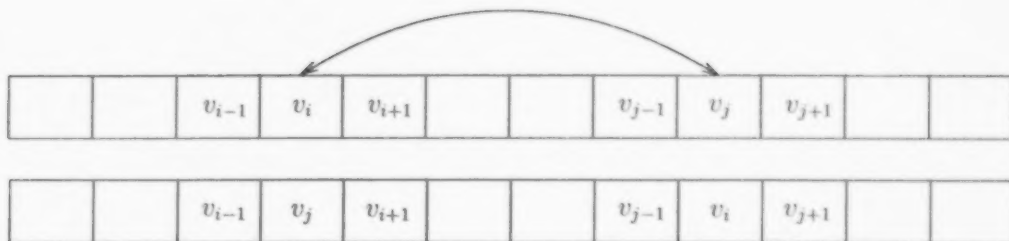


Figure 2: Exchange of type 2

Even for moderate values of n , these exchanges lead to large neighbourhoods: in an instance with n cars, one must consider $O(n^2)$ potential insertions and swaps. Since n is usually large (e.g., several hundreds), a sampling strategy is used to reduce the computational burden associated with the evaluation of the solutions in the neighbourhood $N(s)$. At each iteration only a subset of all potential exchanges is considered. Every pair of positions (i, j) is in fact considered every η iterations, where $\eta = \min\{n, 200\}$ in our implementation. This is accomplished as follows: each pair (i, j) is randomly assigned an integer μ_{ij} in $[0, \eta - 1]$ and, at each iteration t , the pair (i, j) is considered for a move only if $\mu_{ij} = t \pmod{\eta}$. The μ_{ij} values are themselves randomly regenerated every η iterations.

Finally, for exchanges of type 2, only those moves yielding a better solution are considered. In other words, no swap is allowed when a local minimum is reached. This rule prevents the search process from performing a long series of swaps leaving the objective function value unchanged.

2.2.3 Tabu tenures

Whenever the car occupying position i is moved to a different position (exchange of type 1), reinserting it in its previous position is forbidden for θ iterations, where $\theta = 200$ in our implementation. Similarly, whenever two cars are swapped (exchange of type 2), reinserting them in their original positions is forbidden for θ iterations. The tabu status associated with the insertion of car v in position i can be overridden through a so-called attribute based aspiration criterion if allowing the insertion would let the search reach a solution with a smaller cost than that of the best solution found with this attribute. This is accomplished by maintaining, for each car v and each position i , a value λ_{vi} representing the cost of the best solution found during the search in which car v appeared in position i . In the case of a swap, the tabu status is lifted only if the aspiration criterion is met for both of the associated insertions, i.e., the cost of the solution is smaller than both $\lambda_{v_i, j}$ and $\lambda_{v_j, i}$.

2.2.4 Continuous diversification

The search procedure uses a continuous diversification mechanism. Whenever a type I move results in a deterioration of the current solution, the value of $g(s)$ is artificially increased to $g(s) = g(s) + h(s) + p(s)$, where $h(s)$ is a diversification term penalizing frequently performed moves and $p(s)$ encourages the creation of long runs of cars of the same colour, while satisfying the limit ℓ . For a given move of the car v_i to position j , we set $h(s)$ equal to $\sqrt{n}$ multiplied by one of the following values:

1. the number of times a car has moved from position i to position j ;
2. the number of times car v_i has moved from its current position to another position.

The algorithm alternates between these two values each time a new tabu search process is initiated in Step 3b. The term $h(s)$ is used to direct the search toward less explored regions of the solution space when a local optimum is reached.

The term $p(s)$ is introduced to encourage the creation of long groups of cars of the same colour. If v_i and v_j have the same colour, but v_i belongs to a smaller group of consecutive cars of the same colour than v_j , then $p(s) = 0$. The same is true if v_i and v_{j+1} have the same colour, but v_i belongs to a smaller group of consecutive cars of the same colour than v_{j+1} . In both cases, moving v_i from its current position to position j will lead to the creation of a larger group of cars of the same colour. In all other cases, one sets $p(s) = \alpha\psi$, where α is the coefficient associated to the number of colour changes in the objective function, and $\psi = 0.1$ in our implementation. We should point out that this term is considered only when it is theoretically possible to reduce the number of groups of cars of a given colour.

2.2.5 Stopping criterion

The tabu search stops when no improvement in the value of $f(s)$ has been obtained for κ consecutive iterations, where $\kappa = 200$ in our implementation.

2.3 Perturbation mechanisms

Several mechanisms are used to perturb the current solution $\hat{s}$ and obtain a new solution s' in Step 3a of the iterated tabu search heuristic:

1. *Random swaps*

This perturbation consists of randomly selecting χ pairs of positions (i, j) with $i \neq j$ and swapping the cars v_i and v_j . We use $\chi = 3$ in our implementation.

2. *Random shuffling within a subsequence*

With this perturbation, a subsequence of τ consecutive cars is randomly selected and the corresponding cars are reordered at random to produce a new solution. We use $\tau = 6$ in our implementation.

3. *Mirror transformation of a subsequence*

A subsequence of τ consecutive cars is selected and the order of the corresponding cars is reversed, as shown in Figure 3.

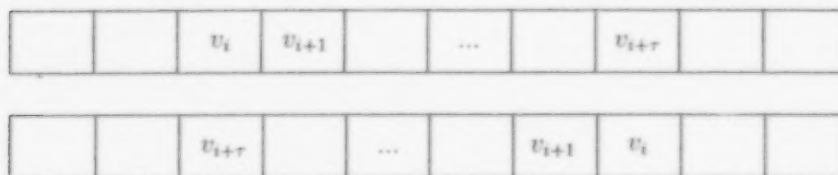


Figure 3: Mirror transformation of the subsequence $(v_i, \dots, v_{i+\tau})$

4. *Random move of a subsequence*

This perturbation randomly selects a subsequence $(v_i, \dots, v_{i+\tau-1})$ of τ consecutive cars and a position j outside this subsequence. The subsequence is then inserted between positions j and $j + 1$.

5. *Removal and reinsertion of a subsequence of cars of the same colour*

In this perturbation, a position k with $1 \leq k \leq n$ is first chosen at random. A group of cars of the same colour is then formed by identifying all consecutive cars in positions $\dots, k - 2, k - 1, k + 1, k + 2, \dots$ with the same colour c_{v_k} as v_k . A position i outside this group is then selected at random and the first position $j \geq i$ such that $c_{v_j} \neq c_{v_{j+1}}$ is identified. Finally, the group of cars is inserted between positions j and $j + 1$.

6. *Application of all improving swaps and of a proportion π of neutral swaps*

This perturbation consists of applying the following procedure for each position i with $1 \leq i \leq n - 1$: for every position $j \geq i + 1$, compute the impact of swapping the cars in positions i and j ; if the swap reduces the cost it is applied; otherwise, if it leaves the cost unchanged and the cars v_i and v_j are not identical, it is applied with probability π . Two cars are said to be identical if they are of the same colour and possess the same features. We should mention that because only a fraction $1/\eta$ of the neighbourhood

$N(s)$ is examined at each iteration of the tabu search heuristic, improving swaps may be identified when performing this perturbation. We use $\pi = 0.1$ in our implementation.

Every time one of the above six perturbations is applied in Step 3a of the ITS, the weights α , β and γ used in the objective function are also modified in the hope of further diversifying the search. In addition to the original weight configuration $f_1 = (10^6, 10^3, 1)$, we also consider two different configurations: $f_2 = (10^4, 10^2, 1)$ and $f_3 = (10^4, 10^4, 1)$. The following sequence is used to alternate between the different configurations: $(f_1, f_2, f_1, f_3, f_1, f_2, \dots)$.

The objective function is also modified according to the elapsed CPU time since the beginning of the search. The search is indeed divided into three phases of equal duration (i.e., each uses one third of the total allotted CPU time). In the first phase, only the most important term in the objective function (i.e., the one with the highest weight in configuration f_1) is considered. In the second phase, the second most important term also enters the objective function. Finally, the last phase considers all three terms. There are nevertheless two exceptions to this rule: 1) if the most important term is the number of colour changes, then the first phase is skipped and the second phase is twice as long; 2) if PRC violations are the most important term and the search reaches a solution with no violation in the first phase, then the search immediately switches to the second phase (the same applies to NPRC violations in the second phase).

2.4 Acceptance criterion

If the weight configurations f_1 or f_2 are used, the solution $\tilde{s}$ obtained at the end of Step 3b is accepted and replaces the current solution $\hat{s}$ if its cost $f(\tilde{s})$ exceeds that of the best known solution s^* by at most one unit. If the weight configuration f_3 is used, the solution $\tilde{s}$ is accepted if it is better than s^* with respect to the second objective. In the latter case, however, the solution does not replace $\hat{s}$. Indeed, the next perturbation will be applied to $\tilde{s}$ but if the solution obtained at the end of the following intensification phase (with weight configuration f_1) does not satisfy the acceptance criterion, the search will return to $\hat{s}$.

2.5 Neighbourhood evaluation

An important consideration in the implementation of our algorithm is the fast evaluation of the solutions in the neighbourhood $N(s)$ of the current solution s . Because there are many solutions to evaluate at each iteration and each of them is obtained through a small modification of the current one, it is worth maintaining some data structures to avoid explicitly evaluating the objective function value $f(s)$ for every solution in $N(s)$.

To this end, we use two special data structures which are updated every time a move is performed on the current solution. First, we compute, for each position i , the number of consecutive cars of the same colour as v_i that immediately precede or follow position i . Taking this information into account, one can directly evaluate the impact of an insertion or swap on the number of colour changes and constraint violations.

Second, we compute, for each position i and feature r , the total increase in ratio constraint violations that would result from the addition in position i of feature r . This value must take into consideration every subsequence of p_r consecutive cars containing position i . This information is of course irrelevant if the car v_i currently in position i possesses feature r . Similarly, we also compute the total impact of removing feature r from position i . Again, this information will be irrelevant if the car currently in position i does not possess feature r . Using this information, one can measure in constant time the impact of most swaps. Furthermore, one can directly identify many cases where insertions have no impact on ratio constraint violations and thus no computations are necessary. Special care must nevertheless be taken when evaluating insertions and swaps involving positions that give rise to overlapping subsequences of length p_r for any feature $r \in R$.

3 Computational Experiments

We have tested our heuristic on three instance sets provided by Renault. The first two sets (A and B) were given to the participants of the 2005 ROADEF Challenge to help them test their algorithm and tune its parameters. The last set (X) was used to rank the teams in the final phase. The three sets comprise a total of 80 instances. All experiments were performed on a 1.7 MHz Pentium 5 computer and, unless otherwise indicated, ten minutes of CPU time were allotted to the solution of each instance.

To evaluate the usefulness of each type of perturbation, we have first tested our heuristic by considering all types of perturbations, and by then successively disregarding one type at a time. For comparison purposes, we have also implemented a variant without any form of perturbation. Table 1 reports the scores obtained with the nine variants of the heuristic on the 16 instances from set A. The *score* is computed by dividing the *deviation* by the *range*. The deviation is the difference between the value of the solution produced by the algorithm and the worst solution found during all experiments, while the range is the difference between the worst and the best solution values identified during these experiments. The score is a normalized measure of solution quality which necessarily lies between 0 and 1. It also corresponds to the formula used by the organizers of the ROADEF Challenge for ranking the teams.

The first three columns of Table 1 indicate the instance number, the score obtained without any form of perturbation (column *None*), and the score obtained with all types of perturbation (column *All*). In the latter case, each of the six types of perturbation has an equal probability of being selected whenever a perturbation must be performed in Step 3a of the algorithm. In addition, the weight configuration is modified as explained at the end of Section 2.3. The next seven columns indicate the score obtained with a variant of the heuristic obtained by considering all types of perturbation except one (the one indicated as the column name): 1. random swaps; 2. random shuffling; 3. mirror transformation; 4. random move of a subsequence (block); 5. removal and reinsertion of a subsequence of the same colour; and 6. neutral swaps. Again, equal probabilities are associated with the types of perturbation being considered.

Table 1: Impact of perturbation operators on the score

No.	None	All	Weights	Swaps	Shuffle	Mirror	Block	Colour	Neutral
1	0	0.9998	1	0.9998	0.9998	0.9998	0.9998	0.9998	0.9998
2	0	1	1	1	1	1	1	0.9997	1
3	0.0475	0.9544	0.6216	0.7171	0.1953	0.5747	0.7649	0.1942	0
4	0.4211	0.8246	0.7729	1	0.8948	0.8776	0.8073	0.8425	0.4908
5	0	0.5433	0.6265	0.5431	0.4602	0.6685	0.3352	0.7933	0.8733
6	0.4038	0.8299	0.766	1	0.8937	0.8086	0.9579	0.7452	0.3187
7	0	0.5108	0.266	0.7556	0.5921	0.4906	0.4083	0.3069	0.2245
8	0	0.6925	0.7654	0.7775	0.7658	0.8702	0.8713	1	0.4935
9	0	1	0.9975	0.7502	0.7489	0.7484	0.7504	0.998	0.7427
10	0.1733	0.5333	1	0.96	0.7733	0.5333	0.2667	0.72	0.4933
11	0	0.778	0.7505	0.7228	0.9999	0.6395	0.9445	0.8337	0.1943
12	0	0.6059	0.9355	0.871	0.8706	0.5358	0.8686	0.5364	0.1983
13	0.999	1	0.6663	0.9997	0.6667	0.667	0.6667	0.9997	0.3323
14	1	1	1	1	1	1	1	0	1
15	0	0.9996	0.9997	0.7142	0.9996	0.9995	0.9999	0.7139	0.9993
16	0	1	1	1	1	1	1	1	1
Total	3.0447	13.2722	13.168	13.811	12.8607	12.4136	12.6414	11.6833	9.3608

From this table, one can observe that perturbations constitute a fundamental element of our heuristic. Without any form of perturbation, a total score of only 3.0447 is obtained whereas the inclusion of all forms of perturbation yields a score of 13.2722. The marginal contribution of each type of perturbation is, however, limited. With the exception of neutral swaps, the removal of any type of perturbation from the heuristic has a very limited impact on solution quality. Since the results are very similar, we have used for the full version in all remaining experiments.

To make a better use of the different types of perturbation, we have then implemented a roulette-wheel selection mechanism which randomly selects perturbation operators according

to their past performance. To this end, we keep track during the search of how often each type of perturbation has led to an improved solution in Step 3c of the algorithm. The better the performance of a perturbation type, the higher its probability of being selected. More precisely, let ρ_k represent the number of times perturbation k has led to an improved solution, divided by the number of times this perturbation has been performed. The probability of choosing perturbation k is then equal to $\rho_k / \sum_{k=1}^6 \rho_k$. Initially, we set $\rho_k = 1$ for every perturbation. These values are then updated every time a perturbation is applied. In Table 2, we compare the initial implementation (in which each perturbation type has an equal probability of being selected) to the new roulette-wheel selection mechanism. One can see that the total score obtained with the latter variant is significantly higher than that obtained with the initial one. It is also higher than those obtained with any other variant (see Table 1). We have therefore used this variant in the remainder of our experiments.

Table 2: Impact of roulette-wheel selection on the score

No.	Initial	Roulette
1	0.9998	1
2	1	0.9999
3	0.9544	0.718
4	0.8246	0.9823
5	0.5433	0.6254
6	0.8299	0.937
7	0.5108	0.8778
8	0.6925	0.8135
9	1	0.9993
10	0.5333	0.36
11	0.778	0.9447
12	0.6059	1
13	1	1
14	1	1
15	0.9996	1
16	1	1
Total	13.2722	14.2578

We have then performed a series of experiments to fine-tune the various parameters used in the heuristic. In particular, we have sequentially studied the impact of the following values: the length τ used in several types of perturbation, the number κ of iterations allowed without improvement in the tabu search, and the percentage π of neutral swaps performed in perturbations of type 6. In Table 3, we compare the scores obtained with four different values of the parameter τ . When performing these experiments, we have also set $\chi = \tau/2$. These results show that $\tau = 6$ yields the best average performance. In Table 4, we report

similar results for five different values of κ between 25 and 300. The last column of this table indicates the results obtained with a variant in which κ is initially set equal to 25 and is increased by five each time the tabu search is called without any improvement, until it reaches 300. The value of κ is reset to 25 whenever an improved solution is identified. These results show that $\kappa = 200$ offers the best compromise when considering all instances. Finally, Table 5 shows comparative results for three values of π . The algorithm is somewhat sensitive to this parameter and its best value appears to be 0.1 on the 16 test instances.

Table 3: Impact of parameter τ on the score

No.	Value of τ			
	4	6	8	10
1	0.8002	1	1	1
2	0.9999	0.9999	1	1
3	0.7143	0.718	0.9539	0.8577
4	0.5439	0.9823	0.4379	0.5605
5	0.8339	0.6254	0.5831	0.6673
6	0.2979	0.937	0.5734	0.5953
7	0.7147	0.8778	0.4698	0.7144
8	0.5966	0.8135	0.549	0.6603
9	0.7445	0.9993	0.996	0.9945
10	0.6	0.36	0.4667	0.6933
11	0.9723	0.9447	0.9725	1
12	0.4043	1	0.4697	0.2039
13	0.6653	1	0.6667	0.332
14	1	1	1	1
15	0.9994	1	0.9996	0.7137
16	1	1	1	1
Total	11.8871	14.2578	12.1384	11.993

We have finally compared the results obtained by the best variant of our heuristic to those obtained by the other teams who took part in final stage of the competition. In Tables 6, 7 and 8 we report the worst and best solution values obtained by any of the teams. We then indicate the solution value obtained by our heuristic, followed by its score. Here, the score is computed with respect to the difference between the worst and best solutions reported by the teams, as was done by the organizers. These tables also indicate the full name of each instance from each set.

For the 16 instances from set A, we obtain a total score of 15.4036 out of a maximum of 16, and our heuristic identifies the best known solution five times. With the exception of the tenth instance for which we obtain a score of 0.4815, our scores are always above 0.96. For the tenth instance, the range is very small and any deviation from the best solution results

Table 4: Impact of parameter κ on the score

No.	Value of κ					
	25	50	100	200	300	25-300-5
1	1	0.9998	0.9998	1	0.9998	0.9998
2	1	1	1	0.9999	1	1
3	1	0.9518	0.8578	0.718	0.8578	0.8584
4	0	0.2801	0.2799	0.9823	0.2799	0.5261
5	0.7499	1	0.6257	0.6254	0.6257	0.6673
6	0	0.4681	0.4679	0.937	0.4679	0.425
7	0.51	1	0.4696	0.8778	0.4696	0.9187
8	0.4431	0.6386	0.6604	0.8135	0.6604	0.5458
9	0.9935	0.992	0.7427	0.9993	0.7427	0.7477
10	0	0.2267	0.76	0.36	0.76	0.4267
11	0.8058	0.9445	0.9999	0.9447	0.9999	0.8338
12	0.2715	0.4694	0.3375	1	0.3375	0.3367
13	0.0003	0.6653	0.9997	1	0.9997	0
14	1	1	1	1	1	1
15	0.9996	0.9996	0.9995	1	0.9995	0.9996
16	1	1	1	1	1	1
Total	9.7738	12.6359	12.2004	14.2578	12.2004	11.2856

in a low score. For the 45 instances of set B, we obtain a total score of 44.1166. Again, we obtain excellent results for most instances. The lowest score is 0.7952 and the heuristic finds the best solution 16 times. Finally, Table 8 reports the results for the set X which was used in the final stage of the ROADEF Challenge. Here, instead of executing the heuristic only once as we did for sets A and B, we executed the heuristic five times as was done for the ranking of the teams in the competition. The value reported in column *Result* is thus the average over five runs. This time, the results are even closer to the best known solution, with a total score of 18.8112 on a maximum of 19. The best solution was identified eight times and the worst score is 0.8881.

The ranking method used for the ROADEF Challenge consisted of computing the score for each instance as we explained above. The final score of a team was then set equal to the sum (or average) of all scores. However, the final ranking of the teams differs from the ranking based on these scores. Indeed, the jury reckoned that the difference between the scores were too small among the best teams and that the scores of the best candidates were biased by those of the worst ones. Thus, an iterative procedure was used to eliminate the last candidate at each step of the ranking process until only three teams remained. This procedure corrected the observed bias but also had the effect of disadvantaging the teams that favoured a robust and stable program over a more aggressive but less reliable one.

Table 5: Impact of parameter π on the score

No.	Value of π		
	0.05	0.1	0.15
1	0.9998	1	0.8002
2	1	0.9999	1
3	0.9528	0.718	0.6194
4	0.4378	0.9823	0.6832
5	0.7505	0.6254	0.5426
6	0.4465	0.937	0.5534
7	0.2046	0.8778	0.6534
8	0.5497	0.8135	0.3696
9	0.7465	0.9993	0.9933
10	0.68	0.36	0.44
11	0.6669	0.9447	0.8611
12	0.271	1	0.6017
13	0.334	1	0.6663
14	1	1	1
15	0.9992	1	0.9994
16	1	1	1
Total	11.0392	14.2578	11.7834

Indeed, with the new ranking procedure, the score of a feasible solution with a cost slightly higher than the best one may be the same as that of an infeasible solution (i.e., 0). For comparison purposes, we report in Table 9 the final ranking published by the organizers, the total score of each team and the number of infeasible solutions produced by each method. This table shows that the heuristic we had submitted for the competition actually achieved the third best score (18.6665) while our improved implementation obtains a slightly better performance with a score of 18.8112. In any case, the difference with the score of the winning team (equal to 18.9935) is rather small.

To evaluate the performance of our heuristic in time-critical situations, we have finally performed two additional experiments. First, we have computed the time needed to obtain the first feasible solution during the search. For each of the 80 instances from sets A, B and X, we could always find a feasible solution within less than one second. In fact, the longest time before reaching feasibility (0.11 second) occurred for an instance of set B with 1270 cars, 6 PRCs and 7 NPRCs. Second, we have computed the score associated with the best solution found after only 30 seconds of CPU time. The corresponding results are reported in Table 10 and show that the solutions identified after 30 seconds are close to the best solutions with a total score of 18.1148 on the 19 instances from set X.

Table 6: Comparative results on instances from set A

Instance	Worst	Best	Result	Score
022-3-4-EP-RAF-ENP	112001	31001	31001	1
022-3-4-RAF-EP-ENP	11048003	11039001	11039002	0.9999
024-38-3-EP-ENP-RAF	71118491	4000302	4017352	0.9997
024-38-3-EP-RAF-ENP	63376290	4249083	4293209	0.9993
024-38-5-EP-ENP-RAF	82165438	4034309	4070379	0.9995
024-38-5-EP-RAF-ENP	75477143	4280079	4322099	0.9994
025-38-1-EP-ENP-RAF	1704363	99720	156774	0.9644
025-38-1-EP-RAF-ENP	320753	231134	234603	0.9613
039-38-4-EP-RAF-ch1	84248000	13129000	13149000	0.9997
039-38-4-RAF-EP-ch1	68344000	68155000	68253000	0.4815
048-39-1-EP-ENP-RAF	27135374	61290	78308	0.9994
048-39-1-EP-RAF-ENP	27202853	174612	188737	0.9995
064-38-2-EP-RAF-ENP-ch1	240741	112759	112759	1
064-38-2-EP-RAF-ENP-ch2	54014	34051	34051	1
064-38-2-RAF-EP-ENP-ch1	64499822	63423782	63423782	1
064-38-2-RAF-EP-ENP-ch2	27376087	27367052	27367052	1
Total				15.4036

4 Conclusions

This paper has introduced a simple yet general iterated tabu search algorithm for the CSP. This algorithm does not rely on the availability of a feasible initial solution and can start from any car permutation. The algorithm is robust in the sense that it generates feasible solutions within less than one second on test instances with more than one thousand cars. Although it has obtained slightly worse solutions than those produced by the best algorithms proposed in the ROADEF Challenge, it quickly yields high quality solutions whose values are very close to those of the best known solutions. It is thus well adapted to situations where robustness and simplicity are required. This algorithm can also be used to provide feasible initial solutions to more sophisticated algorithms.

Acknowledgments

We thank the members of the 2005 ROADEF Challenge organizing committee and, in particular, Van-Dat Cung and Alain Nguyen. This work was partly supported by a Strategic research grant provided by HEC Montréal, and by the Canadian Natural Science and Engineering Research Council under grants 227837-04 and OGP0039682. This support is gratefully acknowledged.

References

- M. Delaval, E. Castelain, and J.C. Gentina. Dynamic resequencing of cars on assembly lines using simulated annealing. In *Proceedings of the 28th International Symposium on Automotive Technology and Automation, Dedicated Conference on Lean/Agile Manufacturing in the Automotive Industries*, pages 169–175, Stuttgart, Germany, 1995.
- C. Gagné, M. Gravel, and W.L. Price. Solving real car sequencing problems with ant-colony optimization. *European Journal of Operational Research*, 2005. Forthcoming.
- J. Gottlieb, M. Puchta, and C. Solnon. A study of greedy, local search and ant colony optimization approaches for car sequencing problems. In S. Cagnoni et al., editors, *EvoWorkshops 2003, Lecture Notes in Computer Science 2611*, pages 246–257. Springer, Berlin, 2003.
- M. Gravel, C. Gagné, and W.L. Price. Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society*, 56: 1287–1295, 2005.
- K.S. Hindi and G. Ploszajski. Formulation and solution of a sequencing problem in car manufacture. *Computers & Industrial Engineering*, 26:203–211, 1994.
- T. Kis. On the complexity of the car sequencing problem. *Operations Research Letters*, 32: 331–335, 2004.
- H.R. Lourenço, O.C. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer, Boston, 2002.
- A. Misevicius. Using iterated tabu search for the traveling salesman problem. *Information Technology and Control*, 3(32):29–40, 2004.
- E. Muhl, P. Charpentier, and F. Chaxel. Optimization of physical flows in an automotive manufacturing plant: Some experiments and issues. *Engineering Applications of Artificial Intelligence*, 16:293–305, 2003.
- A. Nguyen. Challenge ROADEF 2005: Car sequencing problem. Technical report, Renault, France, 2005.
- L. Perron and P. Shaw. Combining forces to solve the car sequencing problem. In J.-C. Régim and M. Rueher, editors, *CPAIOR 2004, Lecture Notes in Computer Science 3011*, pages 225–239. Springer, Berlin, 2004.

- S.G. Ponnambalam, P. Aravindan, and M.S. Rao. Genetic algorithms for sequencing problems in mixed model assembly lines. *Computers & Industrial Engineering*, 45:669–690, 2003.
- K. Smith, M. Palaniswami, and M. Krishnamoorthy. Traditional heuristic versus Hopfield neural network approaches to a car sequencing problem. *European Journal of Operational Research*, 93:300–316, 1996.
- C. Solnon. Solving permutation constraint satisfaction problems with artificial ants. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 118–122. IOS Press, Amsterdam, 2000.

Table 7: Comparative results on instances from set B

Instance	Worst	Best	Result	Score
022-EP-ENP-RAF-S22-J1	14125	3109	3120	0.999
022-EP-RAF-ENP-S22-J1	52130	19144	20145	0.9697
022-RAF-EP-ENP-S22-J1	13023160	13022148	13022150	0.998
023-EP-ENP-RAF-S23-J3	147018462	48000316	48000324	1
023-EP-RAF-ENP-S23-J3	126384071	48310008	48313006	1
023-RAF-EP-ENP-S23-J3	54349063	51327031	51338028	0.9964
024-V2-EP-ENP-RAF-S22-J1	1209127491	1074850430	1084910438	0.9251
024-V2-EP-RAF-ENP-S22-J1	1222366438	1074299068	1088344255	0.9051
024-V2-RAF-EP-ENP-S22-J1	135253163	134023158	134213456	0.8453
025-EP-ENP-RAF-S22-J3	4096795	3912479	3912530	0.9997
025-EP-RAF-ENP-S22-J3	317852	172180	185133	0.9111
025-RAF-EP-ENP-S22-J3	133129840	126127589	126132219	0.9993
028-ch1-EP-ENP-RAF-S22-J2	375048139	54003076	54027092	0.9999
028-ch1-EP-RAF-ENP-S22-J2	54078415	54049124	54055124	0.7952
028-ch1-RAF-EP-ENP-S22-J2	39103295	38098188	38098227	1
028-ch2-EP-ENP-RAF-S23-J3	70010	70006	70006	1
038-ch2-EP-RAF-ENP-S23-J3	6073	4071	4071	1
028-ch2-RAF-EP-ENP-S23-J3	6000072	4000071	4000071	1
029-EP-ENP-RAF-S21-J6	42150325	37150167	37150170	1
029-EP-RAF-ENP-S21-J6	35307150	35167170	35173150	0.9573
029-RAF-EP-ENP-S21-J6	53773436	52711171	52717171	0.9944
035-ch1-EP-ENP-RAF-S22-J3	67061061	67052049	67052053	0.9996
035-ch1-EP-RAF-ENP-S22-J3	67053060	67036061	67036064	0.9998
035-ch1-RAF-EP-ENP-S22-J3	6162093	6156090	6156092	0.9997
035-ch2-EP-ENP-RAF-S22-J3	388350188	385341205	385341205	1
035-ch2-EP-RAF-ENP-S22-J3	385191367	385187351	385187351	1
035-ch2-RAF-EP-ENP-S22-J3	7654671	7651671	7651671	1
039-ch1-EP-ENP-RAF-S22-J4	52777	29117	29163	0.9981
039-ch1-EP-RAF-ENP-S22-J4	106155	78089	78145	0.998
039-ch1-RAF-EP-ENP-S22-J4	68044265	55045096	55045163	1
039-ch3-EP-ENP-RAF-S22-J4	864	197	199	0.997
039-ch3-EP-RAF-ENP-S22-J4	211317	189103	189127	0.9989
039-ch3-RAF-EP-ENP-S22-J4	59230127	59214671	59214681	0.9994
048-ch1-EP-ENP-RAF-S22-J3	19049547	200	207	1
048-ch1-EP-RAF-ENP-S22-J3	233461	161378	165414	0.944
048-ch1-RAF-EP-ENP-S22-J3	65146959	64115670	64123692	0.9922
048-ch2-EP-ENP-RAF-S22-J3	3026446	3000337	3000344	0.9997
048-ch2-EP-RAF-ENP-S22-J3	3150967	3094029	3097024	0.9474
048-ch2-RAF-EP-ENP-S22-J3	58317217	58283180	58283180	1
064-ch1-EP-ENP-RAF-S22-J3	17277	182	206	0.9986
064-ch1-EP-RAF-ENP-S22-J3	234094	130187	135211	0.9516
064-ch1-RAF-EP-ENP-S22-J3	63128588	62095288	62098336	0.9971
064-ch2-EP-ENP-RAF-S22-J4	69339	69130	69130	1
064-ch2-EP-RAF-ENP-S22-J4	143133	130069	130069	1
064-ch2-RAF-EP-ENP-S22-J4	32054181	31052178	31052178	1
Total				44.1166

Table 8: Comparative results on instances from set X

Instance	Worst	Best	Result	Score
022-RAF-EP-ENP-S49-J2	12202004	12002003	12002003	1
023-EP-RAF-ENP-S49-J2	314044	192466	206073	0.8881
024-EP-RAF-ENP-S49-J2	49844834	337006	400415	0.9987
025-EP-ENP-RAF-S49-J1	16950509	160408	160828	1
028-CH1-EP-ENP-RAF-S50-J	338608118	36341495	36386296	0.9999
028-CH2-EP-ENP-RAF-S51-J	9403	3	3	1
029-EP-RAF-ENP-S49-J5	181024	110298	111242	0.9867
034-VP-EP-RAF-ENP-S51-J1	145530	55995	59060	0.9658
034-VU-EP-RAF-ENP-S51-J1	12103052	8087036	8095256	0.998
035-CH1-RAF-EP-S50-J4	6010000	5010000	5010000	1
035-CH2-RAF-EP-S50-J4	7056000	6056000	6056000	1
039-CH1-EP-RAF-ENP-S49-J	88237	69239	69311	0.9962
039-CH3-EP-RAF-ENP-S49-J	285101	231030	231671	0.9882
048-CH1-EP-RAF-ENP-S50-J	2410707	197006	197825	0.9996
048-CH2-EP-RAF-ENP-S49-J	34951538	31077916	31096860	0.9951
064-CH1-EP-RAF-ENP-S49-J	67263120	61187230	61218289	0.9949
064-CH2-EP-RAF-ENP-S49-J	72400	37000	37000	1
655-CH1-EP-RAF-ENP-S51-J	53000	30000	30000	1
655-CH2-EP-RAF-ENP-S52-J	306032000	153034000	153034000	1
Total				18.8112

Table 9: Ranking of the teams

Team	Final ranking	Score	Infeasible solutions
Nouioua et al.	1	18.9935	0
Ribeiro et al.	2	18.8896	0
Naddef et al.	3	17.9173	1
Bloemen	4	17.9465	1
Kuipers	5	17.834	1
Gavranovic	6	16.8067	2
Cordeau et al.	7	18.6665	0
Stützle et al.	8	16.9372	0
Gendreau et al.	9	18.1654	0
Pawlak et al.	10	16.3762	0
Gravel et al.	11	17.6204	1
Benoist et al.	12	16.419	1
Jaskiewicz et al.	13	16.8937	1
Montemanni	14	12.6513	0
Bonizzoni et al.	15	11.8235	0
Caseau	16	13.865	1
Zufferey et al.	17	13.7882	3
Klau et al.	18	8.7213	2
Our new heuristic	–	18.8112	0

Table 10: Scores with 30 seconds of CPU time

Instance	Result	Score
1	12002003	1
2	222873	0.7499
3	2243030	0.9615
4	218238	0.9966
5	39027504	0.9911
6	3	1
7	17865	0.893
8	74838	0.7895
9	8098864	0.9971
10	5010000	1
11	6056000	1
12	70634	0.9266
13	238496	0.8619
14	202259	0.9976
15	31108837	0.992
16	61236494	0.9919
17	38200	0.9661
18	30000	1
19	153035800	1
Total		18.1148

